Evaluating the Privacy Guarantees of Location Proximity Services

GEORGE ARGYROS, THEOFILOS PETSIOS, SUPHANNEE SIVAKORN,
and ANGELOS D. KEROMYTIS, Network Security Lab, Columbia University
JASON POLAKIS, Computer Science Department, University of Illinois at Chicago

Location-based services have become an integral part of everyday life. To address the privacy issues that emerge from the use and sharing of location information, social networks and smartphone applications have adopted location proximity schemes as a means of balancing user privacy with utility. Unfortunately, despite the extensive academic literature on this topic, the schemes that large service providers have adopted are not always designed or implemented correctly, rendering users vulnerable to location-disclosure attacks. Such attacks have recently received major publicity as, in some cases, they even exposed citizens of oppressive regimes to life-threatening risks. In this article, we systematically assess the defenses that popular location-based services and mobile applications deploy to guard against adversaries seeking to identify a user's location. We provide the theoretical foundations for formalizing the privacy guarantees of currently adopted proximity models, design practical attacks for each case, and prove tight bounds on the number of queries required for carrying out successful attacks in practice.

To evaluate the completeness of our approach, we conduct extensive experiments against popular services including Facebook, Foursquare, and Grindr. Our results demonstrate that, even though the aforementioned services implement various privacy-preserving techniques to protect their users, they are still vulnerable to attacks. In particular, we are able to pinpoint Facebook users within 5m of their exact location. For Foursquare and Grindr, users are pinpointed within 15m of their location in 90% of the cases, even with the strictest privacy settings enabled. Our attacks are highly efficient and complete within a few seconds. The severity of our findings was acknowledged by Facebook and Foursquare, both of which have followed our recommendations and adopted our design of a safe proximity scheme in their production systems. As the number of mobile applications offering location functionality will continue to increase, service providers and software developers must be able to assess the privacy guarantees that their services offer. To that end, we discuss viable defenses that can be currently adopted by all major services, and provide an open-source testing framework to be used by researchers and service providers who wish to evaluate the privacy-preserving properties of applications offering proximity functionality.

CCS Concepts: • **Information systems** → **Location-based services**; *Social networks*; • **Security and privacy** → **Social network security and privacy**; *Privacy protections;*

Additional Key Words and Phrases: Location-based services, location privacy, location proximity, user discovery attacks, spatial cloaking

This work is supported by the National Science Foundation, under grant CNS-13-18415.
Authors' addresses: G. Argyros, T. Petsios, S. Sivakorn, and A. D. Keromytis, Computer Science Department, Columbia University in the City of New York, 1214 Amsterdam Avenue, New York, NY 10027; emails: {argyros, theofilos, suphannee, angelos}@cs.columbia.edu; J. Polakis, Department of Computer Science, College of Engineering, University of Illinois at Chicago, 851 S. Morgan St. MC 152, Chicago, IL 60607; email: polakis@uic.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2017 ACM 2471-2566/2017/02-ART12 $15.00
DOI: http://dx.doi.org/10.1145/3007209

ACM Reference Format:
George Argyros, Theofilos Petsios, Suphannee Sivakorn, Angelos D. Keromytis, and Jason Polakis. 2017. Evaluating the privacy guarantees of location proximity services. ACM Trans. Priv. Secur. 19, 4, Article 12 (February 2017), 31 pages.
DOI: http://dx.doi.org/10.1145/3007209

1. INTRODUCTION

In an era in which the use of mobile devices has become an integral part of everyday life, social networks and mobile applications are providing users with an increasingly rich set of location-based features. However, apart from the benefits it offers, this functionality also exposes users to a diverse set of threats, ranging from the inference of sensitive data [Minami and Borisov 2010] (e.g., medical issues, political inclination and religious beliefs), to physical risks such as stalking [Scheck 2010]. Moreover, recent articles have revealed that the threat landscape of location privacy is very complex: government agencies have been massively surveilling social network users employing programs such as PRISM [Greenwald and MacAskill 2013] and law enforcement agencies have been targeting users employing fake social network accounts, in order to gain access to their personal data and track their whereabouts by monitoring their check-inbehavior [Lardner 2010; Police Forum 2013].

Revealing a user's location is a significant privacy breach [Zhao et al. 2013]; thus, services are adopting the more privacy-oriented approach of *location proximity*: notifying users about who is nearby, and at what distance, without revealing the actual location. However, when the exact distance to a user is revealed by the service, trilateration attacks become feasible. A plethora of such examples has been presented in the media recently, the most notable being that of the Egyptian government using trilateration to locate and imprison users of gay dating apps [Paton 2014; Noack 2014].

It is becoming apparent that preserving the locational privacy of users is an urgent matter; in some cases, it may even be a matter of life and death. Such alarming incidents have caught the attention of popular services that, in turn, have deployed defense mechanisms to prevent localization attacks. Although the academic community has made extensive efforts to address the problem of location privacy [Zhong et al. 2007; Šikšnys et al. 2009; Shokri et al. 2011; Narayanan et al. 2011], services offering location-based functionality have been slow to adopt the proposed solutions.

In this article, we explore the privacy guarantees of 10 popular social networks and location-based services (LBSs). We audit these services and identify the defense mechanisms that they deploy to protect the location privacy of their users. To evaluate the various defenses that have been adopted by the industry, we formalize the problem of locating users as a search problem in the discrete Euclidean plane. To our knowledge, this is the first formal treatment of user discovery attacks in LBSs. We prove tight bounds on the number of queries required to attack a service under different proximity models, and devise optimal algorithms that realize those attacks. The lower bounds on the query complexity of our techniques provide useful insight on the effectiveness of common mitigations against localization attacks, such as rate limiting the number of queries that users can issue.

We evaluate our attacks against four of the audited services that employ a diverse set of countermeasures: Facebook, Grindr, Skout, and Foursquare's Swarm. We show that user discovery attacks against proximity services may require complex techniques; our attacks include geometric algorithms that gradually reduce the candidate bounding area where a user resides, the employment of colluding accounts for obtaining side-channel information on the distance between users, and the utilization of statistical algorithms for coping with the randomization used by services as a defense mechanism. Our results demonstrate that, despite the defense mechanisms in place, attacks are still possible and practical for use at scale and on a continuous basis (real-time tracking).

In particular, assuming only that the attacker and victim are connected in the social network, we pinpoint Facebook users within 5m of their actual location in 3s, and 90% of Swarm users within 15m in 7s, with each service having its strictest privacy settings. We even stress-test our attacks and demonstrate the feasibility of tracking moving targets in real time. Due to the recent events involving Grindr [Noack 2014], the service hides the distance information for citizens of oppressive regimes. Even without any distance information disclosed, we are able to carry out successful attacks by inferring the distance to our target. Using a pair of colluding accounts, and the distance-based ordering of users by Grindr, we pinpoint 67% of the users within 10m of their exact location, and 98% within 19m. Similarly, even though Skout implements a sophisticated randomization defense, we are able to pinpoint its users within 37.4m, on average.

Our findings reveal that there is no industry standard for ensuring the locational privacy of users; most of the current defenses are based on ad-hoc approaches that often exhibit a lack of understanding of the technical intricacies of localization attacks. Despite the active effort to prevent such threats, every service that we audited was vulnerable to at least one of our attacks. To provide a robust solution, we revisit an obfuscation mechanism from the literature, spatial cloaking [Gruteser and Grunwald 2003], and apply it to the domain of distance-based proximity services. By quantizing the plane and mapping users to points on a grid, the service can prevent adversaries from pinpointing users to a finer precision than that of a grid cell. To incentivize services to adopt this defense, we provide a precise characterization of both the privacy obtained (under certain assumptions), and the trade-off between privacy and usability. *After the disclosure of our findings, Facebook and Foursquare acknowledged the severity of our attacks and, following our guidelines, adopted spatial cloaking for protecting their users.*

The main contributions of this article are the following:

—We present a formal treatment of user discovery attacks within a proximity service. We model the problem, prove the lower bounds on the query complexity, and design algorithms that match the corresponding lower bounds.
—We evaluate the privacy of popular proximity services through extensive experimentation, and reveal the shortcomings of existing proximity models and defenses. The disclosure of our findings to the services resulted in Facebook and Foursquare adopting spatial cloaking. In addition, we evaluate the accuracy of our attacks in Facebook and Foursquare after the adoption of spatial cloaking.
—We analyze the practical aspects of our attacks, and identify key characteristics that affect their performance and accuracy. We provide guidelines for impairing the attacks and ensuring a minimum level of privacy without incurring a significant deterioration of the quality of service.
—We release an open-source auditing framework for assisting developers and researchers in assessing the privacy of proximity services. Our framework has already been used by Facebook for evaluating their newly adopted spatial cloaking mechanism.

1.1. Responsible Disclosure

Disclosing attacks against services raises ethical issues as, one might argue, adversaries may have previously lacked the know-how to conduct such attacks. However, recent events demonstrated that adversaries are already targeting users. While the attacks seen in the wild were straightforward, and preventable by certain existing defenses, it is crucial to assess the privacy guarantees of popular services and prevent future attacks. The false sense of security cultivated by the services and major media due to misconceptions of the privacy offered [Constine 2014a, 2014b; Frizell 2014] and flawed recommendations (e.g., rounding distances [Wardle 2014]), expose users to life-threatening situations. We contacted the services that we evaluated, and provided

them with a detailed analysis of our attacks, our guidelines for preventing them, and an analysis on the privacy/usability trade-off of our spatial cloaking approach. As a result of our disclosure, Foursquare and Facebook have adopted spatial cloaking. Furthermore, the Facebook security team used our testing framework for evaluating their cloaking mechanism.

Document Structure: The rest of the article is organized as follows. Section 2 presents the threat model and relevant information on location proximity. Section 3 provides a formal treatment of the user discovery problem. Section 4 describes practical aspects of the attacks, and Section 5 presents an overview of popular proximity services and their respective characteristics. We experimentally evaluate our algorithms against actual services in Section 6. In Section 7, we set guidelines for implementing proximity functionality, and in Section 8 we describe our open-source auditing framework. Related work is discussed in Section 9 and we present our conclusions in Section 10.

2. BACKGROUND AND THREAT MODEL

Proximity Disclosure. The distance between two users can be disclosed in various forms. For simplicity, we refer to the adversary as Mallory and the target as Bob.

Exact distance: This is the most trivial case to attack, as a trilateration algorithm can identify Bob's exact location with very high accuracy. Such algorithms have been explored extensively (e.g., Thomas and Ros [2005]).

Location disks: The service returns an upper bound of the distance, creating a disk centered at Mallory that encompasses Bob (e.g., "Bob is less than 500m away").

Location rings: The service returns a range as the distance information, creating a ring (two concentric circles) centered at Mallory, with Bob located between the two circles. This may be the result of a simple rounding function (e.g., distances between 51m and 99m are rounded to 100m).

Quantized distances: the plane is quantized with a grid, creating cloaking regions that obfuscate the user's location. Proximity can be disclosed in two forms: (i) users are mapped to grid points, and distances are calculated from their grid points; or (ii) proximity is Boolean and denotes co-location within the same cell.

Threat Model. The adversary can be any entity interested in determining a user's location—a government or law enforcement agency conducting user surveillance ([Lardner 2010; Police Forum 2013]), a third party (e.g., insurance company) interested in inferring private data, or a malicious individual (e.g., stalker) [Scheck 2010]. To highlight the inefficiency for existing designs and countermeasures, we adopt a weak adversarial model; the adversary uses only the distance information revealed by the system.

Our attacks *do not require prior knowledge of the user's whereabouts*, and the only requirement is to have an account in the service to obtain some type of information about the distance to the user. In Section 6, we demonstrate that we can identify a user's location with high precision, and track a moving target in real time.

3. MODELLING DISCOVERY ATTACKS

In this section, we provide the theoretical modelling of our user discovery attacks. We formulate our problem as a search problem in the discrete Euclidean plane. This is justified by the fact that both services and protocols (e.g., GPS) cannot provide arbitrary accuracy. By modelling it as a discrete problem, we can adapt the size of the input to match the accuracy provided by the service. For the following, we will follow the notation found in standard algorithmic textbooks [Cormen et al. 2001]. In addition, we use the term k-approximation for an optimization problem to denote an algorithm that outputs a solution with size at most k times the size of the optimal solution.

We consider a target user u residing at a point p_u of the discrete Euclidean plane. The attacker can request proximity information regarding the location of the user u. This is obtained through an oracle, which we refer to as a *proximity oracle* P. Since the attacker can fake her own location, she can query the proximity oracle from any point within the Euclidean plane. Thus, the proximity oracle accepts a point p and returns proximity information for the point p and the location p_u of the target user. We denote by $P_u(\cdot)$ the proximity oracle that, for an input of a point p, outputs some function of p, p_u. Also, we define as $\text{dist}(p_1, p_2)$ the Euclidean distance between two points p_1, p_2. We proceed to define the user discovery problem, our main algorithmic problem, in the context of location proximity services.

Definition 3.1 (*User Discovery Problem* (*UDP*)). Let p_u be a point in the discrete Euclidean plane and A an area containing p_u. In the UDP, the goal is to identify the point p_u, given as input the area A and black box access to a proximity oracle P_u.

In the following sections, we will describe three different implementations of the proximity oracle that capture the protocols used by real services. For each of these oracles, we describe how to solve the UDP given access to the respective oracle.

3.1. Disk User Discovery Problem

We start by giving the definition of the first oracle.

Definition 3.2 (*Disk Proximity Oracle*). A disk proximity oracle $P_{r,u}(p)$ with radius r accepts as input a point p in the discrete Euclidean plane and is defined as

$$P_{r,u}(p) = \begin{cases} 1 & \text{if } \text{dist}(p, p_u) \leq r \\ 0 & \text{otherwise.} \end{cases}$$

This model captures services and protocols that inform the user as to whether another user is within a certain distance of his current location; otherwise, the user is not in proximity and no further information is given. We define the Disk User Discovery Problem (DUDP) to be the UDP given black box access to a Disk Proximity Oracle. We solve the DUDP by partitioning the problem into two subproblems, which require a different approach in order to be solved. First, we wish to restrict the user within a single disk of radius r and, second, to search that disk for the target point p_u.

In the former subproblem, the user is given a possibly large area A, which she wants to cover with disks of radius r in order to restrict the search area within a single disk. We call this problem the Disk Coverage Problem. To achieve an efficient attack, we wish to cover the area with the minimum number of disks.

Definition 3.3. In the **Disk Coverage Problem**, the input is an area A in the discrete Euclidean plane and a number $r > 0$. The goal is to cover the area A with the minimum number of disks of radius r.

Essentially, in the context of LBSs, and knowing that the target user's location is restricted within a single disk of radius r, we seek to perform the minimum number of queries to ensure that we have covered A completely, and either located the user within a particular disk or determined that the user is not located within A. If the user is located within A, her location is restricted within a particular disk of radius r, and one has to use the proximity oracle to further refine the user's location up to a single point. We call this subproblem the Disk Search Problem.

Definition 3.4. In the **Disk Search Problem**, the input is a single disk of radius r along with a proximity oracle $P_{r,u}(\cdot)$. The goal is to uniquely pinpoint the point p_u within the input disk.

Note that the Disk Search Problem is exactly the DUDP when the input area is restricted to a disk of radius r. Because the two cases are handled in a different manner, we address them separately. Next, we examine each subproblem and describe algorithms for solving them.

Solving Disk Coverage. To generalize our attack, we assume that the only information that the attacker has is a very coarse-grained approximation of the location of the targeted user; for example, Mallory might know which state Bob lives in. Given a total area in which the user might reside, our first goal is to to pinpoint the user within a disk of radius r, as provided by the proximity oracle.

A problem that corresponds precisely to the Disk Coverage Problem is the Minimum Dominating Set (MDS) problem in a special class of graphs called Unit Disk Graphs (UDGs). In the MDS problem, one is given as input a graph $G = (V, E)$ and the goal is to find a set $D \subseteq V$ such that, for every $v \in V$, there exists a $u \in D$ for which $(u, v) \in E$. UDGs are a special class of geometric graphs; even though a number of equivalent definitions exist, we will use what is referred to as the proximity model [Clark et al. 1990]:

Definition 3.5 (*Proximity Model for UDG*). Consider a set P of n points in the plane. The **UDG** for the points in P is the undirected graph $G = (P, E)$, where P is the set of n points and, for every $u, v \in P$, we have that $(u, v) \in E$ if and only if $\text{dist}(u, v) \leq k$, for some predefined $k > 0$.

The MDS problem in UDGs has been studied extensively. While it is still NP-hard [Masuyama et al. 1981] to find an exact solution, the approximation ratio that has been achieved is much better than for the general MDS problem in which it is NP-hard to find a solution smaller than $O(\log n)$ times the optimal solution. Specifically, there exists a Polynomial Time Approximation Scheme (PTAS) [Nieberg and Hurink 2006] whose complexity is impractical for the problem instances that we aim to tackle, as well as a 5-approximation in linear time [Marathe et al. 1995] and a 3-approximation in time $O(n^{18})$ [De et al. 2011].

Note that the areas that one needs to cover might spread across entire cities or even larger areas. Thus, the respective graphs may contain millions of nodes. At that scale, even algorithms with quadratic complexity may result in prohibitively large runtimes. Fortunately, for the MDS problem, there exists an algorithm with a runtime linear to the size of the input set and computing a 5-approximation of the optimal solution.

The algorithm works by taking an arbitrary node from the graph, adding it to the dominating set, removing all neighboring nodes that are already covered by the chosen node, and repeating the process until the graph is empty. It is easy to see that the algorithm will run in time $O(|V|)$. If the graph is a UDG, the algorithm provides a covering with a size at most 5 times the minimum covering [Marathe et al. 1995]. Since this is a very general construction, there are many ways in which one could chose the set of points while satisfying the requirements of the algorithm. Although these different constructions would provide the same worst-case approximation ratio, they might perform very differently in practice. In our algorithm, we create the points of our set by tiling the 2-dimensional plane with hexagons, a common technique for ensuring coverage with minimum node placement in cellular networks [Wang et al. 2005]. Each hexagon edge is equal to $\sqrt{3}r$, where r is the radius for the look-up granularity. Note now that, since no two points in our set are at a distance less than r, our algorithm creates a 5-approximation of the minimum covering, and choosing the points takes $O(|V|)$, on average.

Solving Disk Search. After the user is restricted to a disk of radius r, the goal is to refine the location to a single point. We present an algorithm that solves the Disk

Table I. Notation Used for the DiskSearch(·) Algorithm

S	Input set of points to DiskSearch(·)
$R(S)$	Circumscribed rectangle of set S
k	Length of the short edge of $R(S)$
d	Length of the long edge of $R(S)$
$I_{R(S)}$	Set of points within $R(S)$
$C_r(p)$	A disk centered at point p with radius r
p_m	Middle point of the short edge of $R(S)$
l_m	Line parallel to the long edge of $R(S)$ and crossing p_m
S_1	$S \cap C_r(p)$
S_2	$S \backslash S_1$

Search Problem using $O(\log r)$ queries. One can easily show that the result is optimal. Our algorithm works like a binary search algorithm, using the oracle to cut down the candidate points that the user resides in by half with each query. Note that we might not always be able to split the candidate points into two equals sets in every cut; however, as we will show, the overall number of queries required is still $O(\log r)$.

We start by providing some definitions that will be used throughout the description of the algorithm: for a given set S of points in the Euclidean space, we denote with $R(S)$ the circumscribed rectangle that includes those points, and by $I_{R(S)}$ the points in the rectangle $R(S)$. The two edges of $R(S)$ are referred to according to their lengths, "short edge" and "long edge," with lengths k, d, where $k \leq d$. For a point p, we denote with $C_r(p)$ the points in the disk centered at p with a radius r. We say that the algorithm performs a cut on a set S, resulting in two subsets S_1, S_2, when the proximity oracle $P_{r,u}$ is queried on a point p and $S_1 = S \cap C_r(p)$, $S_2 = S \backslash S_1$. We denote the middle point in the short edge of $R(S)$ by p_m. Finally, let l_m be a line parallel to the long edge of $R(S)$ passing through p_m. Table I provides an easy reference of our notations.

The algorithm DiskSearch(S) proceeds recursively:

(1) For input of a set S, if $|S| = 1$, return $p \in S$.
(2) For each point p_i in l_m, starting at a distance r from $R(S)$, and moving toward $R(S)$, check the size of the set $S \cap C_r(p_i)$. If for a point p_i it holds that $|S \cap C_r(p_i)| > |S|/2$, then set $p = p_{i-1}$.
(3) Make a call to the proximity oracle on point p:
—If $P_{r,u}(p) = 1$, recurse on S_1.
—If $P_{r,u}(p) = 0$, recurse on S_2.

The correctness of the algorithm is evident once we bound the number of recursive calls that the algorithm makes. For the runtime analysis, we assume that the calls to $P_{r,u}(\cdot)$ require constant time. We prove the following theorem.

THEOREM 3.6. *The* DiskSearch(·) *algorithm has a time complexity of* $O(r \log r)$*, and will do at most* $O(\log r)$ *queries to the proximity oracle* $P_{r,u}(\cdot)$*.*

Moreover, we prove the following theorem showing the optimality of our algorithm.

THEOREM 3.7. *Let* $P_{r,u}(\cdot)$*, a proximity oracle with radius* r*, and* A *be a set of points in the discrete Euclidean plane. Denote by* OPT_{MDS} *the minimum size of a dominating set for the UDG spawned from the points in* A*. Then, any deterministic algorithm solving the DUDP in the set* A *will have to make* $\Omega(OPT_{MDS} + \log r)$ *queries to the disk proximity oracle* $P_{r,u}(\cdot)$ *in the worst case.*

In the context of LBSs, A represents all legal coordinates for which a query can be performed toward the service. In the case in which no restriction is imposed by the

service and any pair of coordinates is accepted, one may choose an arbitrary precision quantization up to the decimal precision supported by the service.

3.2. Rounding User Discovery Problem

Now, we turn to a different oracle implementation. This proximity oracle model computes the distance between p and p_u but rounds the resulting value up to some predefined accuracy. To correctly define the oracle in this model, we define the notion of a rounding class.

Definition 3.8 (*Rounding Class*). For an interval $I_k = (a, b) \subseteq \mathbb{R}_+$, we define the rounding class R_k to be the tuple (I_k, δ_k), where $\delta_k \in \mathbb{R}_+$ is a rounding value.

Some services (e.g., Facebook) apply different rounding depending on the distance between two users. This motivates the notion of a rounding class family, which we define later to capture this behavior.

Definition 3.9 (*Rounding Class Family*). We define a family of rounding classes $S = \{(I_1, \delta_1), \ldots, (I_n, \delta_n)\}$ to be a set of rounding classes such that the following hold:

(1) The set $I = \{I_1, \ldots, I_n\}$ forms a partition of $\mathbb{R}_+$.
(2) For i, j, we have that $i < j \Rightarrow \delta_i < \delta_j$.
(3) For $k > 1$, it holds that $\delta_k \leq \inf I_k$.

The first condition implies that, for every possible distance, there is a corresponding rounding value. The second condition asserts that the rounding values monotonically decrease as the distance decreases. The third condition intuitively states that the rounding added is small compared to the actual distance. Conditions 2 and 3 are not necessary for our attacks to work; however, they are natural constraints that we found in all services, and simplify the description and analysis of our algorithms.

Next, we define the operator $\lceil x \rceil_\delta$ that rounds x to the closest number $x' \geq x$ such that $\delta \mid x'$[1]. Note that one can similarly define the operators $\lfloor \cdot \rceil_\delta$ and $\lfloor \cdot \rfloor_\delta$. When attacking real services, we consider rounding classes that also use these operators. However, the algorithms themselves do not change in any way. For simplicity, we will use the $\lceil \cdot \rceil_\delta$ operator throughout the presentation and analysis.

Definition 3.10. A **Rounding Proximity Oracle** $P_{S,u}(p)$ indexed by a family of rounding classes S is defined as:

Let $(I_k, \delta_k) \in S$ be a rounding class such that $\text{dist}(p, p_u) \in I_k$. Then, $P_{S,u}(p) = \lceil \text{dist}(p, p_u) \rceil_{\delta_k}$.

We define the Rounding User Discovery Problem (RUDP) as the UDP given access to a Rounding Proximity Oracle. Our problem definition also covers other models. For example, an oracle that outputs exact distances is a rounding proximity oracle with the rounding class family $S = \{(\mathbb{R}_+, \delta)\}, \delta \to 0$.

Solving the RUDP. Intuitively, if we have the exact distance to the target user, a simple trilateration will give us his location. However, in our case, the rounding operation is applied, which adds noise to the result. Thus, the result of applying a trilateration on a rounded distance is to reduce the area where the user resides. Next, we exploit the fact that, for each rounding class $R_k = (I_k, \delta_k)$, we have that $\delta_k \leq \inf I_k$. Reapplying the trilateration algorithm within the reduced area will result in getting smaller rounding errors, thus, further reducing the search area. This procedure is repeated until we have reached $R_1 = (I_1, \delta_1)$, in which case, we use the rounding value

[1]Since δ might not be an integer, we abuse the notation here to denote by $\delta \mid x$ that x is an integer multiple of δ.

ALGORITHM 1: Algorithm for the RUDP

Input: Rounding Proximity Oracle $P_{S,u}(\cdot)$, Area A;
Output: Location of the target user in the Euclidean plane;
$\texttt{Area}_0 = A$;
$i = 1$;
$l = |S|$;
while $i < l \wedge |\texttt{Area}_i| > 1$ **do**
  $\texttt{Area}_i = \texttt{Trilaterate}(\texttt{Area}_{i-1}, P_{S,u}(\cdot))$;
  $i = i + 1$;
end
$P_{r,u} = \texttt{SimulateDiskProximityOracle}(P_{S,u}(\cdot))$;
$p_u = \texttt{DiskSearch}(\text{Area}_k, P_{r,u}(\cdot))$;

of R_1 to define a disk proximity oracle and execute the $\texttt{DiskSearch}(\cdot)$ algorithm that we presented. Specifically, we define the disk proximity oracle $P_{r,u}(\cdot)$ for a radius $r = \delta_1$ as follows:

$$P_{r,u}(p) = \begin{cases} 1 & \text{if } P_{S,u}(p) \leq \delta_1 \\ 0 & \text{Otherwise} \end{cases}$$

Note that, after the last trilateration, the remaining points to search are located in an area of size δ_1^2. Therefore, a disk of radius δ_1 is large enough to run the $\texttt{DiskSearch}(\cdot)$ algorithm. Algorithm 1 shows the implementation of the attack. We define a procedure $B = \texttt{Trilaterate}(A, P_{S,u}(\cdot))$, which accepts as input an area A and a rounding proximity oracle and, by querying $P_{S,u}(\cdot)$, performs a trilateration to reduce the possible area in which the user is residing to B. The details of the trilateration algorithm are known [Huang and Narayanan 2014] and thus omitted; finally, the procedure $\texttt{SimulateDiskProximityOracle}$ creates a disk proximity oracle as described before.

Analysis. The iterative application of trilateration is guaranteed to reduce the search area into a smaller rounding class every time, because we know that, for a rounding class R_k, we have that $\delta_k \leq \inf I_k$. Albeit natural, this restriction is not necessary; if at any point the algorithm cannot guarantee that trilateration will provide a reduced search area, then we can define the proximity oracle similar to the way we defined it earlier, then execute the $\texttt{DiskSearch}(\cdot)$ algorithm. The number of queries that the algorithm makes to the oracle is $3 \cdot |S| + O(\log \delta_1)$, where δ_1 is the rounding value of the smaller rounding class R_1. Additionally, the time required by the trilateration procedure is also constant; thus, the total runtime is $O(|S| + \delta_1 \log \delta_1)$.

We prove the following theorem establishing a matching lower bound on the complexity of the RUDP problem for deterministic algorithms.

THEOREM 3.11. *Any deterministic algorithm solving RU DP given access to a proximity oracle $P_{S,u}$ indexed by a family of rounding classes $S = ((I_1, \delta_1), \ldots, (I_n, \delta_n))$ requires $\Omega(|S| + \log_{\delta_1})$ queries to $P_{S,u}$ in the worst case.*

3.3. Randomized User Discovery Problem

Under this model, the proximity oracle query result for a point follows a probability distribution rather than being a deterministic function of the distance. We capture this using the following definition:

Definition 3.12. A randomized proximity oracle $P_{\mathcal{D},u}(p)$ indexed by a probability distribution $\mathcal{D}$ is defined as follows:

$$P_{\mathcal{D},u}(p) \approx \mathcal{D}(p_u, p).$$

In other words, each point queried to the oracle will get its value from a probability distribution $\mathcal{D}$ based on the point queried and the location of the target user.

We define the randomized user discovery (RANDUDP) to be the UDP given access to a randomized proximity oracle. We also assume that the distribution $\mathcal{D}$ is known. Although it is unlikely in practice for a service to publish the distribution used, an attacker can deploy a fake victim user with a known location and collect a large number of samples from the distribution in order to create an approximation of $\mathcal{D}$.

Note that the way RANDUDP is defined, there exist certain probability distributions for which it is impossible to locate the targeted user. For example, in the case in which $\mathcal{D}$ is the uniform distribution over $\{0, 1\}$, querying the oracle will not provide any information on the location of the user.

Nevertheless, such distributions will also fail to offer any real functionality for the users of the service; therefore, they are not encountered in practice. Usually, the oracle $P_{\mathcal{D},u}$ will return the correct proximity information with some probability and an incorrect value otherwise.

For the following, we consider an intuitive randomization model that we denote as the *disk search with Gaussian noise*. Under this model, the following randomized proximity oracle is defined.

Definition 3.13. A randomized disk proximity oracle with radius r and standard deviation σ is defined as follows:

$$P_{r,\sigma,u}(p) = (1 - z_{p_u}(p))P_{r,u}(p) + z_{p_u}(p)(1 - P_{r,u}(p)),$$

where $P_{r,u}$ is a disk proximity oracle with radius r and $z_{p_u}(p) \in \{0, 1\}$ is the error function satisfying

$$\Pr[z_{p_u}(p) = 1 | dist(p, p_u) \in [a, b]] = \int_a^b f(x, r, \sigma)dx,$$

where $a, b \in \mathbb{R}_+$ and $f(x, r, \sigma)$ is the PDF of $\mathcal{N}(r, \sigma)$, the normal distribution with mean r and standard deviation σ.

Intuitively, under this model, when an attacker attempts to detect points in distance r from the user in order to perform a cut, an increasing amount of noise is added, forcing the attacker to perform a wrong cut with the DiskSearch algorithm, therefore stopping the attack. Moreover, the noise is reduced as we move away from that "danger zone," thus not affecting significantly the experience of normal users.

Out of all the different defenses that we encountered in our research, this model was the most difficult to crack in the sense that it requires a much larger number of queries than the previous models. Later, we present an algorithm in order to solve the DiskSearch problem with Gaussian noise, and state a reduction showing that the problem of distinguishing between normal distributions with different mean values reduces to this problem.

Solving Disk Search with Gaussian Noise. Here, we derive an algorithm to solve the DiskSearch problem with Gaussian noise. Our algorithm is an application of the Maximum Likelihood Estimation (MLE) principle, which is widely applied in statistics and machine learning. Specifically, the algorithm proceeds as follows:

(1) Given as input an area A, let r' be the radius of the minimum circle surrounding the area A. Then, for a number k, given as input, sample k points randomly in distance $d \in [r - r', r + r']$ from the centroid of A. Let $S = \{(p_1, l_1 = P_{r,\sigma,u}(p_1)), \ldots, (p_k, l_k = P_{r,\sigma,u}(p_k))\}$.

(2) Compute the MLE over the candidate points in A as

$$p_u = \underset{\hat{p}_u \in conv(A)}{\operatorname{argmax}} \sum_{i=1}^{k} \log \Pr[P_{r,\sigma,u}(p_i) = l_i],$$

where $conv(A)$ denotes the convex hull of the area A.

Analysis. For more information regarding the MLE algorithm, we refer the reader to Hastie et al. [2001]. However, we note that the reason behind selecting the points within the convex hull of A is that, since the probability density function of the Gaussian distribution is log-concave, the problem presented earlier can be written as a convex optimization problem and, if we select the points from within the convex hull of A, then one may utilize efficient convex optimization algorithms [Boyd and Vandenberghe 2004] in order to find the point with the maximum likelihood.

4. PRACTICAL DIMENSIONS OF UDP

Here, we describe the practical aspects and challenges of carrying out the attacks against actual services.

Search space. Depending on the adversary's knowledge, the search space may be significantly large. Chaabane et al. [2012] found that 29% of users publicly disclose their current city, while 48% disclose it to their contacts. Such information can potentially also be obtained through a geo-inference attack [Jia et al. 2014]. While our attack is very efficient, collateral information can reduce the size of the search space.

Proximity Oracle Querying. As the attack requires interaction with a service, the following practical aspects become relevant, as they affect the attack.

Connections. Services may restrict which users can obtain proximity information. Dating apps disclose that information to users that are not connected (i.e., strangers). Social networks have more privacy-oriented default settings, and disclose the information to the users' contacts only. While this might seem like an obstacle, the adversary can employ fake accounts to befriend the user. Previous work (e.g., Bilge et al. [2009]) has explored this topic extensively, and demonstrated that users are very likely to accept requests from unknown accounts. For the remainder of the article, we assume that, if required, the adversary is connected to the user within the service. Nonetheless, we minimize the number of attackers; when a social connection is required, we employ a single account.

Detection. As it is trivial to send fake coordinates, services deploy mechanisms for detecting this. To identify the mechanisms and their thresholds, we can follow the approach from Polakis et al. [2013]. By knowing the exact thresholds, we can minimize the duration of the attack while avoiding detection.

Attack Accuracy. Accuracy is unavoidably dependent on the GPS precision of the user's device. In our experiments, distances are calculated from the coordinates reported to the service. Thus, we measure the *exact accuracy* of our attacks. In practice, accuracy may be influenced by the error of the device's GPS. However, smartphones are becoming increasingly accurate, with GPS accuracy being within 10m and researchers demonstrating improvements of 30% [Yang et al. 2014]. Furthermore, the accuracy of our attack is such that GPS errors do not alleviate the threat that targeted users face.

Projection Errors. No map projection perfectly represents all areas, and a given projection is constructed to preserve only a few properties (e.g., shape) of the real geographical area. While the attacks are not dependent on the projected coordinates, the coordinates of the search area should be transformed using the projection most suitable for that area to minimize the error introduced. We use the appropriate equidistant

Table II. Popular Services with Location Proximity

	#	Dst	GPS	Grid	Query	Speed	Rand
Facebook	1-5B	◎	✗	✗	✗	✓	✗
Swarm	5-10M	◯	✓	✗	✗	✓	✓
Grindr	5-10M	⊙ | ⌀	✗	✗	✓	✗	✗
Skout	10-50M	◎	✗	✗	✓	✓	✓
MeetMe	10-50M	◯ | ◎	✗	✓	✗	✓	✗
Lovoo	10-50M	◎	✗	✗	✗	✗	✗
Tinder	10-50M	◎	✗	✗	✗	✓	✗
SayHi	10-50M	⊙	✓	✗	✗	✗	✗
Jaumo	5-10M	◎	✓	✗	✗	✗	✗
HelloWorld	1K-5K	◎	✗	✗	✗	✗	✗

Distance: — exact ⊙ — rings ◎ — disks ◯ — none ⌀

conic projection for the continent in which we are searching, centered at our area of interest.

5. LOCATION PROXIMITY: A CASE STUDY

In this section, we analyze a series of popular services offering proximity functionality to verify how applicable and realistic our attacks are. Our analysis demonstrated that our theoretical formulation of the discovery problem and the respective proximity oracle constructions can successfully model the proximity functionality of existing services.

Table II presents the details on the proximity models of each application. The first column of the table (#) denotes the number of downloads for each app, as reported by Google's Play Store, while the rest of the columns represent, respectively, the following attributes:

—*Dst (Distance information)*: Services may reveal the distance in various forms (e.g., disks, rings, and so on).
—*GPS (GPS coordinates)*: The user's GPS coordinates may be shared with other users; this can be done explicitly (e.g., exact location shown) or implicitly (app receives coordinates of contacts for calculating distance).
—*Grid*: The service may use a grid when calculating distances. In Swarm, while a grid is employed, it does not affect the calculation. In MeetMe, the distance is calculated from the adversary's grid point to the victim's actual location. Thus, neither perform spatial cloaking, which can prevent attacks.
—*Query (Query limiting)*: The service may enforce query rate limiting that prohibits a straightforward execution of the attack. Unless a very strict limit is enforced, one can simply increase the waiting time between queries.
—*Speed (Speed constraints)*: The service may enforce a limit on the speed with which users travel between locations.
—*Rand*: The service adds random noise in the distances returned by the proximity oracle.

After performing this initial analysis, we evaluated our proximity oracle constructions against all possible location disclosure models that we encountered. Toward this end, we selected four applications—Facebook, Swarm, Grindr, and Skout—to examine how our attacks perform against services that reveal distances using rings, disks, or exact location, as well as in cases in which the service performs cloaking or adds random noise to the distances that it reports. The proximity models for the rest of the services fall within one of these categories; as such, the respective analysis is omitted. Before presenting our experimental results in Section 6, we first elaborate more on the characteristics of our selected set of applications.

Swarm is deployed by Foursquare and is built around location proximity: Friends are assigned to location disks of six sizes: (i) 300m, (ii) 1.5km, (iii) 15km, (iv) 30km, (v) 64km, and (vi) everything over 64km. The API used by Swarm has not been made public; as such, we reverse engineer the communication protocol between the client app and the service's servers. Our experiments reveal that Swarm employs a static grid, with users being assigned to their nearest grid point. The grid cells' dimensions are $252m * 274m$. There are two relevant API calls: `UpdateLocation` updates the location and is periodically sent by the app; `Activities_Recent` is sent upon interaction with the app and contains the user's current location. The response contains the active contacts' grid point coordinates and their distance to the sent location. The sent location is not saved, but used only to calculate the distance to the contacts. We use this API call to place our attacker at arbitrary positions. Swarm limits the number of API requests to 500 per hour. Furthermore, speed constraints are enforced only for `UpdateLocation` (we bypass them by using `Activities_Recent`).

Facebook. Two private Graph API calls implement the required functionality: updating the current location and querying for friends in proximity. Our experiments reveal that the coordinates sent with the `nearby-friends` call are not correlated to those received by the `update-location` call, which is considered the user's last known location. While Facebook enforces speed constraints, attackers can exploit this discrepancy to bypass them. Thus, the adversary can carry out the user discovery attack and query the service from multiple vantage points without any speed restriction.

Grindr. Recent articles revealed that the Egyptian government carried out trilateration attacks for locating users of Grindr. To prevent such attacks [Grindr 2014], the new default setting for users located in countries with oppressive regimes is to completely hide the distance information. For other users, exact distances are disclosed. However, the app displays nearby users sorted by distance, regardless of their settings. While this prevents trilateration attacks, Grindr remains vulnerable to our attack. As Grindr is intended for meeting new people, no social connection is required.

Skout. Similar to the previous applications, we reverse engineered the Skout app and identified two API calls that use `SOAP` request and response headers. The first sets a user's location, while the other takes as input a `userID` and retrieves the user's information (including the distance).

6. EXPERIMENTAL EVALUATION

In this section, we present the experimental evaluation of our attack against actual location proximity services. Due to space restrictions, we focus mostly on Swarm, as the system has adopted various defense mechanisms to prohibit adversaries from identifying a user's actual location.

Experimental setup. We deploy a virtual user that plays the role of the targeted user. We know the user's exact location (i.e., the *ground truth*) as it is reported to the service, and use it to calculate the attack's accuracy.

Emergency call geo-location. To assess the accuracy of our attack algorithms, we provide some relevant context; the FCC sets requirements for telephony carriers regarding the geo-location of outdoor calls to emergency lines (e.g., 911). Current revisions are requesting a 50m accuracy within 30s for calls placed indoors from wireless devices [Federal Communications Commission 2015].

6.1. Foursquare—Swarm

We explore if delays are introduced when Bob updates his location. Our experiments show that Mallory sees the change in under 1s, thus enabling real time user tracking.

Projection Error. As the projection error changes depending on the position on the plane, we measure the impact of the projection error across the United States. We

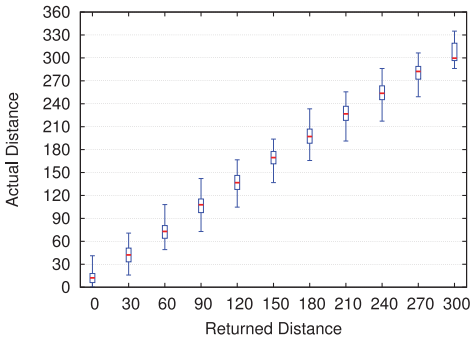

Fig. 1. Rounding distance classes in Swarm.

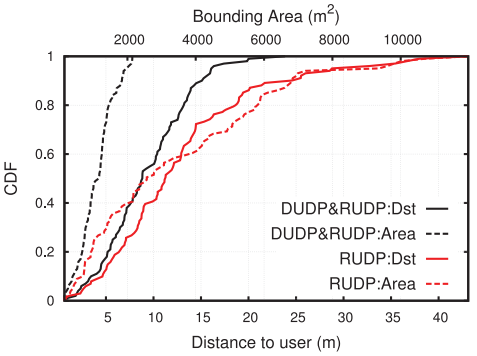

Fig. 2. Accuracy for both attacks against static user in Swarm.

carry out 100 attacks in each state, with Bob placed randomly within the state. When we issue the API call to Foursquare's servers, we receive a response containing the coordinates of the grid point that Bob has been mapped to, and a distance to Bob. In this set of experiments, we consider Bob's location to be the grid point and discard the distance information. Using the coordinates of the grid point, we simulate a disk proximity oracle: we measure the distance from Mallory's location to the grid point and select the corresponding disk size, as shown in the Swarm app (see Section 5). Subsequently, we run our DUDP and `DiskSearch` algorithms to identify Bob's location. In the majority of states, the projection error is negligible; we achieve distances of less than 1m by centering the projection over the state. For 15 states, the distance increases to less than $5m$, and for 5 states to less than $10m$. While more effective projections may exist, these results are sufficiently precise for evaluating our attack algorithms in practice. For the remaining experiments, the victims are placed within New York, where the error is less than 1m.

Complete attack. Next, we evaluate the accuracy of our attack in identifying Bob's exact location and not the grid point to which he has been mapped. As such, we also use the distance information returned. Our experiments reveal that the returned distances are calculated based on Bob's actual location and not the grid point. As can be seen in Figure 1, Swarm returns quantized distances in rounding classes of $30m$. Furthermore, an amount of randomness, which depends on the victim's position within the grid cell, affects the rounding process as an extra measure of privacy. Using our empirical measurements, we translate the returned distance into a location ring.

We follow two attack methodologies. In the first case, we run the Disk User Discovery, Rounding User Discovery, and `DiskSearch` algorithms and achieve maximum accuracy (with an overhead in terms of queries). In the second case, we employ only the Rounding User Discovery and `DiskSearch` algorithms. While the accuracy slightly deteriorates, the attack is more efficient in terms of queries, and can be used in services that apply strict query-limiting or speed constraints.

DUDP&RUDP attack. Figure 2 presents the results from 100 attack experiments carried out in the state of New York using a combination of the DUDP and RUDP algorithms. Apart from being accurate, the attack is also quite efficient, with an average of 55.9 queries and 6.87s required for each attack. In half of the experiments for a static user, the returned location is less than 9m away from the targeted user, while over 98% is less than 20m away. When it comes to the bounding area in which the targeted victim can be found, in 20% of the cases, it is less than $700m^2$. For half of the attacks, it is up to $1{,}160m^2$, and less than $2{,}000m^2$ for 95%. Thus, for 95% of the users, the attack

reduces the bounding area by at least 99.29% compared to the smallest location disk disclosed by the service. If Mallory's goal is to bound the user within a specific area for inferring personal data (e.g., a clinic in proximity to the returned location), she can probabilistically bound the user to an arbitrarily small disk based on these empirical measurements. For example, there is a 50% chance of the user being actually located within a bounding area of $255m^2$ (a disk with a $9m$ radius). Depending on the attack scenario, the adversary can select a different area size, based on the area/probability trade-off.

RUDP attack. Figure 2 presents the results from 100 runs of the resource-efficient attack, which employs only the RUDP algorithm. With this attack, 40% of the users are found less than 10m away, and 95% are less than 30m away. The bounding area is also less accurate, with 22% being less than $1,000m^2$, and 50% less than $2,600m^2$. Thus, for half of the users, the bounding area is reduced by 99%. While not as accurate as the previous attack, this approach is far more efficient, with an average of 18.2 queries and a duration of 2.59s.

Maximum Area. The disk coverage algorithm can cover the entire continental United States with 850 queries, when using disks with $r \simeq 64.3km$ (maximum supported by Swarm). Considering the 500-query limit, one can cover an $\sim$6,174,000km^2 area with 475 queries prior to launching the `DiskSearch`. Thus, the disk coverage attack is very practical, even at a continent level. At a state level, we can pinpoint the victim within a circle of 64km with significantly less queries (e.g., at most, 98 queries for Texas, 64 for California, and 28 for New York).

Moving Targets. We also examine how our attack handles moving targets. The goal is to explore whether our attack can track a moving user in real time. Currently, both Swarm and Facebook update the user's location periodically every few minutes, and upon user interaction with the app. As such, in practice, updates are too infrequent to impact the attack, and the accuracy remains the same. Nonetheless, we set up an experiment to "stress test" our attack, with users moving at various speeds and a location update conducted every 10s. Then, at random intervals, we run our attack against those users, and calculate the distance between the returned location and the user's exact location at that moment in time. Even though this is not a realistic setup, as vehicles in cities are not in continuous motion, but stop intermittently (e.g., traffic lights), it provides a worst-case scenario and demonstrates how users can be effectively tracked, even when in motion.

For targeting the moving victim, we modify the algorithm to provide for the changing location of the victim. Since the user is moving, even if he is bound within a particular area P_i at a certain time t_i, it does not necessarily hold that he will also lie within the same area in a future point at time $t_j > t_i$. If at time t_j we query the oracle for the location of the victim and the response is an annulus A_j, normally, the new polygon in which the victim is bound is $P_j = A_j \cap P_i$. For a moving victim, though, it may hold that $A_j \cap P_i = \emptyset$. In those cases, we set $P_j = A_j \cup P_i$. We do not, however, modify the attack to employ any type of information regarding the user's speed.

Figure 3 and 4 present the results for the accurate (DUDP&RUDP) and efficient (RUDP) attacks, respectively, for users moving with speeds of {10, 20, 40, 60} km/h. As expected, the attack's accuracy deteriorates as the speed increases. Nonetheless, even when the user is moving at 60km/h, the accurate attack achieves a distance less than 10m away for 60% of the experiments, and 85% is up to 20m. The efficient attack pinpoints 60% of the victims within 20m. For users moving at $60km/h$, ten experiments returned a location more than 75m away, due to the user's location being updated during the attack. Depending on the update frequency of the app, the adversary can provide for such cases by running successive attacks within the time window that the location is not updated. Since the goal here is to keep track of the moving user and not infer

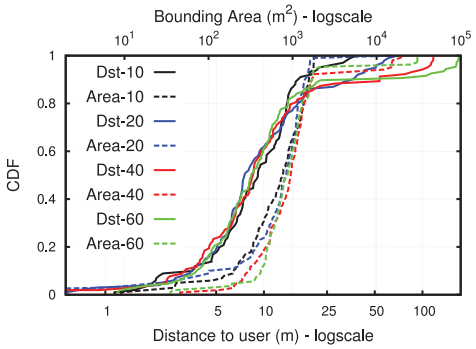

Fig. 3. DUDP&RUDP attack accuracy for moving user in Swarm.

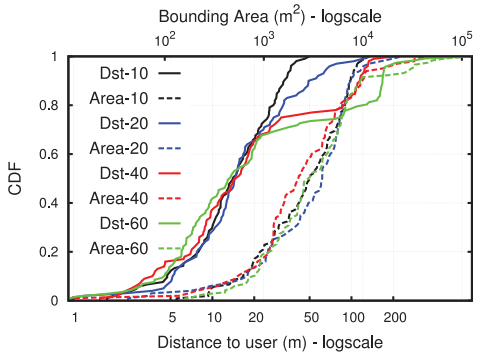

Fig. 4. RUDP attack accuracy for moving user in Swarm.

any sensitive data based on proximity of specific locations, the large bounding areas returned do not affect the objective of the attack.

Overall evaluation. By disclosing the distance from the user's actual position, albeit rounded and with a degree of randomness, the privacy that could be guaranteed due to the grid is lost. While these measures affect the accuracy, the achieved precision is enough for physically locating users or associating them with sensitive locations.

Demonstration. We have a front end for visualizing our attacks. Our back end carries out an attack in real time, and the front end receives the results of each phase and visualizes it at a lower speed. We have uploaded a demonstration[2].

6.2. Facebook

Since Facebook rounds the distance information, we employ the RUDP attack algorithm. We evaluate the effectiveness of our attack by carrying out experiments for static and moving victims. For each experiment, we place the victim accounts randomly in 100 locations in New York. Each attack requires about 20.5 queries, and is completed within 3.05s. As shown in Figure 5, when targeting a static user, the victim in always bounded within an area less than $100m^2$, and the estimated location is less than $5m$ away from the user's exact location. For moving victims, the quality of the attack is significantly reduced for larger speeds: for targets moving at 40km/h, 40% of the users are less than 110m from the approximated location. Due to the attack lasting longer than for Swarm, there is an increased probability of the location being updated during the attack, which results in decreased effectiveness. Nevertheless, adversaries can roughly track a user's movements. Overall, the results of our attack demonstrate that location rounding is ineffective at preserving location privacy, as static users are pinpointed less than 5m from their exact location.

6.3. Grindr

The main observation for conducting the attack is the following: due to the sorting of users, if Mallory knows the location of another user, she can deduce information about the distance to Bob by comparing the colluder's index (position within the user list) to Bob's. As we will show, Mallory is able to leverage this information for defining a *disk proximity oracle* with a radius of her choice. The attack is then reduced to an instance of the disk user discovery problem, and Bob can easily be located. Specifically, Mallory

[2]A demo can be found at: http://youtu.be/7jCr36IT_2s.

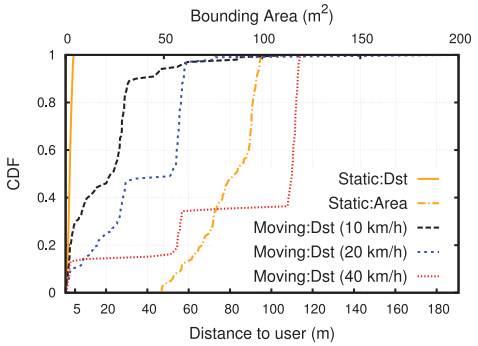

Fig. 5. RUDP attack accuracy for Facebook.

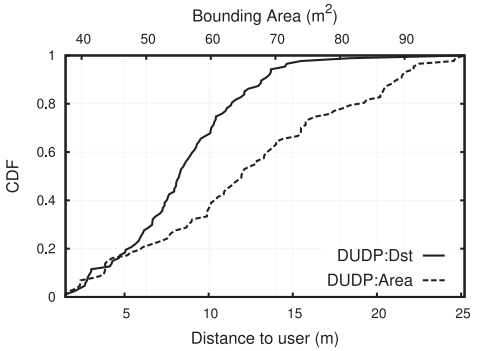

Fig. 6. DUDP attack accuracy for Grindr.

creates two accounts; the attacker and the colluder. Subsequently, she issues a disk proximity oracle query $P_{r,u}(p)$ for a point p, as follows:

(1) With the attacker placed at point p, the colluder is placed at a distance r from p.
(2) The attacker views nearby users in the application. If the targeted user has a smaller index than the colluder, then the oracle replies to the query with 1, otherwise with 0.

Query responses contain the 50 nearest users in ascending order. Due to the large number of users within an entire city, which results in an increased number of requests towards the service for fetching the pages with users from each probing point, we opt for a different attack setup. We run experiments in which Mallory knows that Bob is in a certain area of New York; we set that area to be a disk with a 600m radius, resulting in a search space of $(1.13 * 10^6)m^2$. The results from our experiments can be seen in Figure 6. Again, the attacks are very effective, with 67% of the users located within 10m, 98% within 19m, and, in one case, 25m away. All users are bound within a $100m^2$ area, while the attack requires a total of 58 queries, on average. This includes requesting extra pages when Bob is not found within the first 50 results. In practice, an adversary can deploy real-time attacks in much larger areas by employing multiple accounts. As such, query or speed constraints are ineffective in services for which no social connection is required. As an arbitrary number of accounts can be used, the time required to complete the attack will vary depending on the number of accounts.

This case highlights an important principle of our attacks: as long as the adversary is able to leverage even one bit of information for inferring the targeted user's proximity to a point in the plane, she is able to precisely determine the user's location. This is useful for assessing the privacy guarantees of models that try to conceal the precise distance, but still leak some side channel information about the distance.

6.4. Skout

Skout implements a randomized proximity oracle for defending against user discovery attacks. Previous work [Li et al. 2014] reported that Skout used only quantized distances. Our experiments reveal that the current version has a far more advanced defense mechanism, which adds Gaussian noise to the reported distances, as described in Section 3.3. To evaluate the efficiency of the implemented defense, we deploy our RANDUDP attack. We stress that sampling many points at a close distance for estimating error probability in a specific area does not work, because points in Skout are labeled in clusters, with nearby points getting the same label (either correct or incorrect) with very high probability.

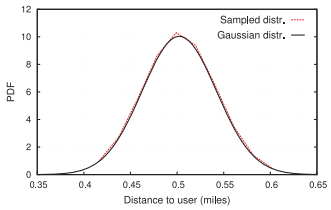

Fig. 7. PDF of Gaussian error distribution in Skout.

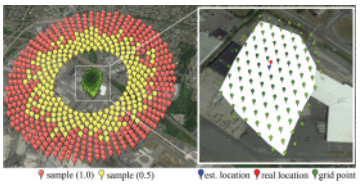

Fig. 8. Instance of RANDUDP attack with sampling points for the MLE calculation.

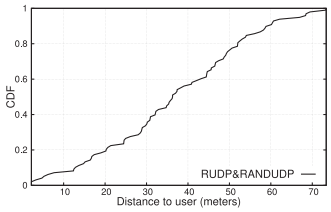

Fig. 9. Attack accuracy for Skout (RUDP & RANDUDP).

In order to estimate the error probability distribution, we employed two accounts and measured the probability of getting an erroneous distance in different locations and distances. In Figure 7, we show the error distribution as a function of the distance and the resulting Gaussian distribution obtained from the samples. In order to fit the sampled distribution in a Gaussian distribution, we used the Levenberg-Marquardt algorithm [Marquardt 1963], from which we obtained a Gaussian with mean $\mu = 0.5$mi and standard deviation $\sigma = 0.039$mi.

Even though Skout uses rounding classes (all distances are rounded up using the $\lceil \cdot \rceil_{0.5}$ operator) by using an RUDP attack, we reduce the candidate area A to an area that can be enclosed in a circle of radius $r' = 0.1$mi. Therefore, the problem effectively reduces to an instance of the disk search problem with Gaussian noise. Then, we execute the algorithm for solving the disk search problem with Gaussian noise, with the parameters described earlier.

Figure 8 presents results for queries between 0.4 and 0.6mi away; red pins depict the points for which Skout returned a distance of 1.0 and a yellow pin to depict those reported as being 0.5mi away. While points that are at most 0.3mi away are consistently rounded up to 0.5, as Mallory enters the range between 0.4 and 0.6mi, the returned distances approximate the Gaussian distribution. The green pins represent candidate points for the target's location, with darker colors denoting higher likelihood. Bob's estimated location is shown with a blue pin and the real location with the red pin.

To evaluate our approach, we perform 100 attacks with the user placed randomly in New York. For the MLE calculation, we use 600 sample points. Each attack requires an average of 1,400 queries, and we use multiple accounts for efficiency. The attack's accuracy is shown in Figure 9; on average, we pinpoint Bob's location within 37.4m despite the advanced defense mechanism employed. Our attack achieves substantially better results than previous attempts [Li et al. 2014], even though Skout now implements a far more sophisticated defense. While this attack requires a much larger number of queries, Skout does not require a social connection for obtaining proximity information. Thus, while this defense would significantly raise the bar in a service that requires a connection (e.g., Facebook), its effect in Skout is diminished, as creating multiple accounts is not a major obstacle.

7. GUIDELINES FOR ENSURING PRIVACY

In this section, we set guidelines for implementing a proximity service that ensures a minimum level of privacy. Based on the theoretical models of our attacks and the experimental evaluation, we identify the characteristics that render existing services or previously proposed mechanisms vulnerable to our attacks. For each mechanism that we propose, we discuss how it affects our attacks, how it holds against certain adversaries, and its impact on the functionality of the service. We set the following assumptions and requirements for the service:

—We assume that the service does not collude with the adversary. Ideally, services would provide private proximity testing, enabling users to test proximity without revealing their location to the service. However, popular services lack the incentives for adopting such an approach.
—In absence of a location verification mechanism, the service should ensure privacy with existing technologies.
—The user's coordinates should never be disclosed to other users (or the app running on their device), and distance calculations must take place on the server.
—We assume that the user does not explicitly reveal his exact location to the adversary. The system cannot protect the user in cases in which the adversary tricks the user into willfully revealing his location.
—The service should ensure a minimum level of privacy. Even against an adversary with many physical resources (e.g., an oppressive regime), the system should ensure that the user's location cannot be bound to an area smaller than a fixed region. The region must be large enough to prevent an adversary from inferring sensitive data.

7.1. A Solution Using Spatial Cloaking - Grid

The services that we audited provide users with some type of *distance* information. Since this is the basis of their operation, any proposed solution should be able to retain this functionality; otherwise, services will probably avoid adopting it, opting for solutions that can provide distance information, even if at the cost of reduced security.

Employing grids for creating cloaking regions has been proposed numerous times in the past. We revisit the concept and analyze its properties based on the characteristics of our attacks. The solution that we present is similar to the proximity protocols presented in Šikšnys et al. [2009], in the sense that it works simply by rounding the coordinates of the user's location, effectively creating a grid over the Euclidean plane. Since only a rounding of the coordinates is required, our solution can easily be implemented and adopted by any proximity service. Here, we present our construction and the trade-offs between privacy and usability offered by the proposed solution.

Grid Construction. Assume that a user is located at a point $p = (x, y)$ in the Euclidean plane. The service uses a predefined rounding value δ and maps the user to the location $p_c = (\lfloor x \rceil_\delta, \lfloor y \rceil_\delta)$. This rounding operation maps all locations within a square grid cell of edge length δ to the centroid of that grid cell. The main questions that we would like to answer for our construction are the following:

(1) *Privacy*: How much privacy does a grid cell of size δ provide to the user?
(2) *Usability*: Does the grid cell construction maintain the usability of the service at an acceptable level?

Privacy. We first notice that the bounding area for a static user inside a grid cell cannot be better than δ^2. The attacker, however, wants to minimize the distance from the target user. Let us first assume that an attacker runs our attack and finds the grid cell of the user; let us also assume that the user is randomly placed at a point p_u within the cell. To minimize her distance from the targeted user, she will place herself at the centroid p_c of the grid cell. Since the accuracy of an attack depends on the distance between the attacker and the user, we would like to estimate the expected value of $\text{dist}(p_u, p_c)$. In the appendix, we prove the following:

$$\mathbb{E}[\text{dist}(p_c, p_u)] \approx 0.382598 \cdot \delta$$

For example, using a grid size of 0.5mi, we can expect that the attacker will be able find the user within a distance of 0.19mi, on average. We believe that the *expected distance* value is an appropriate metric for evaluating the privacy offered by a grid cell

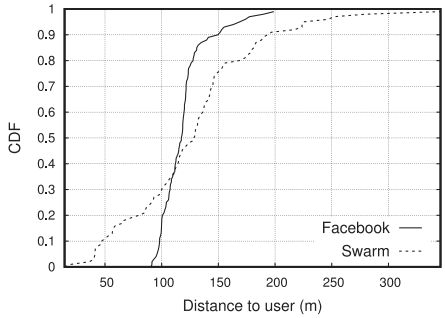

Fig. 10. Effectiveness of the RUDP attack against Facebook and Foursquare after the adoption of spatial cloaking.

of edge length δ, at least in the context of user discovery attacks. Of course, the user will not always be in a random location within a grid cell and social information can be used to further refine his location after the grid cell is detected [Andrés et al. 2013]. Nevertheless, our metric offers an upper bound on the privacy obtained by a grid cell of a certain size.

In the case of private proximity protocols, in which the sender can select the receiver's grid size (e.g., Narayanan et al. [2011] and Zhong et al. [2007]), one must exert caution; unless a safe minimum size is enforced, an adversary can deploy our attacks and bound the user within a small cell that fails to offer any actual protection.

Usability. Another important parameter is quantifying how much this solution will affect the usability of the service. Let us assume that two users u_1, u_2 are mapped to different grid cells p_1, p_2 with a grid edge of length δ. We claim that computing the distances from the rounded coordinates incurs only a small error in the distance computation. With a simple application of the triangle inequality, we have that

$$\text{dist}(p_1, p_2) \geq \text{dist}(\lfloor p_1 \rceil_\delta, \lfloor p_2 \rceil_\delta) - \sqrt{2}\delta$$

$$\text{dist}(p_1, p_2) \leq \text{dist}(\lfloor p_1 \rceil_\delta, \lfloor p_2 \rceil_\delta) + \sqrt{2}\delta.$$

Therefore, the distance $\text{dist}(p_1, p_2)$ is at most $\sqrt{2}\delta$ off from the actual distance.

Trade-off. Now, let us discuss the performance of our solution compared to current implementations, in which rounding is applied to the distance in terms of usability and privacy. As an example, we consider Facebook: the minimum rounding value applied is 0.5mi. Consider now a grid cell of length 0.5mi: as shown previously, a grid size of 0.5mi will provide the user with an average privacy of 0.19mi from our attacks. Moreover, the distance provided to the users will be at most $\sqrt{2} \cdot 0.5 \approx 0.7$mi away from the real distance. Thus, compared to the current implementation, our algorithm will add an additional error of at most 0.2mi to the distance calculations, while providing a much better privacy guarantee. For distances of over 1mi, which are rounded up by 1, our implementation does not incur any additional error to the rounding already added by the service.

Evaluation. As mentioned earlier, after the disclosure of our attacks, Facebook and Foursquare adopted our spatial cloaking approach. We independently verified that the services were no longer vulnerable to our attacks after having followed our guidelines. The results presented in Figure 10 are a sample of 100 attacks for each service, after the adoption of spatial cloaking. Note that the accuracy of the RUDP attack is reduced significantly; in both services, users are located within 100m of their actual

location for less than 30% of the experiments. Recall that, before the adoption of spatial cloaking, Facebook users were always located within 5m of their actual distance, while Foursquare users were located within 30m of their actual distance for 95% of the experiments.

Client-Side Implementation. Another important feature of our proposed solution is that it can be adopted independently by clients and still retain the same guarantees regarding usability and privacy as if the grid was constructed by the service. For example, in Android, various apps exist that allow a user to fake his GPS coordinates. The same apps can be used to generate a rounded version of the actual GPS coordinates, effectively supplying the services with a grid point instead of the real location. Moreover, in "rooted" environments in which more control is given over individual permissions to apps, one could create different grid sizes for each app. Such a solution, would allow a user to maintain a level of privacy while using services with insecure proximity algorithms. Again, we would like to stress that this approach will incur the same exact trade-off between security and usability as a server-side implementation.

7.2. Protecting Moving Users

While a grid ensures privacy down to a certain granularity for static users, maintaining the same level of privacy for moving users becomes complicated. By polling for a user's location, the adversary can infer a user's speed and general direction of movement. Based on that knowledge, and identifying when the user crosses the boundaries of a cell, the user can be bound within an area smaller than the cell. A large body of work has explored methods for protecting [Shokri et al. 2011, 2012; Ghinita et al. 2009; Theodorakopoulos et al. 2014] and attacking [Hashem et al. 2003] the locational privacy of moving users. The solutions vary between different models and assumptions regarding the information available to the attacker and the threat model.

Nevertheless, most of these schemes are built on a common assumption: the existence of an obfuscation mechanism which, given the precise location of the user, generates a pseudolocation. Both defense and attack proposals work under the assumption that such an obfuscation mechanism is employed by the service. As such, our grid proposal is *orthogonal* to the privacy schemes for protecting moving users, as it provides the necessary foundation for deploying those mechanisms. In addition, the formal framework by Shokri et al. [2011] supports our grid construction as an obfuscation mechanism, and can quantify the location privacy of moving users, given a specific Location Privacy Protection Mechanism.

7.3. Location Verification

A crucial aspect of the presented attack is the ability of an adversary to arbitrarily change his position when querying the proximity oracle. By implementing a mechanism that can accurately verify the user's location, the service could significantly impact the attack. Nonetheless, even with such a mechanism in place, an adversary with many resources could carry out the attack at a small scale (e.g., within a small city) by physically deploying agents at various positions and querying the oracle. For example, consider our `DiskSearch` algorithm. This algorithm can be easily executed given two distinct users moving toward the disk, in a line passing from the center on the disk and at orthogonal directions from each other. Then, by issuing frequent queries to the service, the users can pinpoint the targeted user just by moving on their respective line segments. Thus, even in the presence of location verification, our attacks remain practical in certain scenarios.

Previous work has extensively explored methods for the location verification of sensors. While effective solutions have been proposed for specific scenarios, no practical

solution has been found for a web service to verify the location of a smartphone. While proximity verification through location tags [Narayanan et al. 2011] is a promising approach, existing implementations [Zheng et al. 2012; Lin et al. 2012] are too restricting to be deployed in practice. Thus, services must ensure privacy even without such mechanisms in place.

7.4. Attack Mitigation

While the implementation of the grid is straightforward, actual deployment by a service may require significant effort (and time), due to incompatibilities with existing functionality. As such, we also discuss measures that can be trivially (i.e., immediately) implemented by the service and can impact the attack's effectiveness without degrading the quality of service, in lieu of the grid's implementation.

Query rate limiting. A significant aspect of the attack is the feasibility of conducting multiple queries in a small time window. As such, adversaries should be prohibited from flooding the service with location queries for a specific user. Even if the oracle can be queried only once every 5min for a specific user, the attack would require significantly longer time (90min for the efficient attack in Swarm, 100min in Facebook). Applying our lower bounds from Section 3, we get that locating a user in an area of 1km can require up to $2 \log 1000 \approx 20$ queries. A query limit of 5min per query amounts to 100min to locate the user. Of course, this analysis assumes that the user will be randomly placed within this area and no social information is available to the attacker. Nonetheless, even with query limiting, adversaries will be able to identify locations where the use remains for longer periods of time. Obviously, this measure is effective in services for which the adversary can only learn the location of users he is connected to. In services for which no social connection is required, the adversary can use multiple accounts to query about the user; the attack is affected only by the cost of creating accounts. In the face of adversaries with many resources (e.g., an oppressive regime), that cost should not be considered prohibitive.

User movement constraints. This is complementary to the previous measure, as it poses constraints on the timing of the queries issued. If the adversary's user has to remain beneath a certain speed threshold when moving between probing points, the attack duration can be significantly increased. The impact becomes more severe for large search spaces (e.g., large cities). When services do not enforce speed constraints on all proximity-related API calls and expose a call that calculates distances to other users but does not update the previous known location of the user in the system (as is the case with Facebook and Swarm), attackers can leverage those calls for bypassing the constraints and deploying very fast attacks.

Distance disclosure. The feasibility of user discovery attacks within very large search spaces, in a reasonable number of queries, is greatly dependent on the maximum distance disclosed. For services with rounded distances, the adversary can use the intersection of the rings obtained with triangulation. For services that return disk information, if large distances are disclosed (e.g., 10km), the adversary can employ the disk coverage algorithm and greatly reduce the search space with a few queries. The service can mitigate the attack by disclosing proximity up to a certain distance (combined with query and speed constraints) without a loss of functionality. This will significantly impact attacks against large search spaces.

8. AUDITING FRAMEWORK

We have developed a framework designed to facilitate researchers, developers, and privacy-conscious individuals in assessing proximity services. Our principle goal was

to construct a versatile tool for evaluating applications that offer location proximity functionality, with respect to the attacks presented in this work. Our auditing framework (`LBSProximityAuditor`) is implemented in Python and released as an open-source project[3]. The framework provides tests for automatically evaluating common LBS properties like those shown in Table II, and provides an overview of the audited service's ability to effectively guard against location proximity attacks. It also exposes a set of API calls that facilitate common operations on geographical coordinates, both natural and projected. Such operations include calculating intersections or unions of geographic areas, acquiring the coordinates of the borders of polygonal areas, converting from/to `kml` and performing transformations between coordinate systems. Since we cannot predict the characteristics or special requirements of each application being audited, we expect auditors to implement two application-specific calls, one for setting a user's location and one for providing the distance between two users as reported by the audited service. In particular, the auditor is required to implement the following API calls respectively:

```
—auditor_set_location(user, lat, lon)
—auditor_get_distance(user_a, user_b)
```

Once the auditor implements these calls, both the constraints testing and the attacks are expected to work out of the box. However, we assume that the auditor respects our API specification and checks if these calls were successfully executed or not. Moreover, we assume that the auditor has already created an adequate number of accounts to be used in the experiments, and that these accounts are "connected" with each other in the service (if required). Finally, in cases in which the service is not a straightforward application of the RUDP or DUDP proximity oracles (e.g., as was the case with Grindr, in which the oracle is constructed based on the ordering of users), auditors can specify their own proximity oracle. Subsequently, that can be passed as a parameter to our framework for automatically evaluating the service.

9. RELATED WORK

A significant amount of previous work has focused on proposing protocols that enable users to employ an untrusted service without revealing their exact location to it, while achieving desired functionality or being able to detect proximity with other users. Such models are considered out of scope of this work; we do not seek to protect against an adversarial service. This assumption allows for a *lightweight, readily applied* privacy-preserving scheme that protects users against active adversaries but does not require the heavyweight cryptographic operations or complex protocols that are necessary when protecting against a malicious service provider. In the following, we examine related work both from the perspective of the attacks and the defenses presented in this article.

With respect to *Location Proximity*, Zhong et al. [2007] present three private proximity protocols, in which the sender selects the receiver's grid size. Similarly, Narayanan et al. [2011] propose a series of protocols for private proximity testing in which untrusted servers mediate communication. Again, in this scheme, the sender may select the grid's size. This is a common limitation of both works as, using the attacks presented in this article, only the smallest grid size determines the level of offered privacy. Šikšnys et al. [2009] presented FriendLocator, and subsequently VicinityLocator [Šikšnys et al. 2010], which both employ multiple levels of grids of varying size, to calculate proximity while hiding the location from the service. Our attack can be carried out starting at

[3]Repository available at http://www.cs.columbia.edu/nsl/projects/vpsn/.

the level with the largest cells and recursively descending levels until the smallest cell is reached. If the smallest cell is sufficiently large, a minimum level of privacy can be ensured; thus, the complexity of additional cells is not adding to the security of the scheme. Puttaswamy and Zhao [2010] propose moving application functionality to client devices, and treat services and third-party servers as untrusted storage for encrypted data: thus, sensitive information is never disclosed to untrusted parties. A similar solution is "Hide&Crypt," as proposed by Mascetti et al. [2009], in which users store a cloaked version of their location to the service, as well as in their follow-up works "Hide&Seek" and "Hide&Hash" [Mascetti et al. 2011], in which the service is also considered an untrusted hub. Finally, Li et al. [2013] also propose a scheme to detect proximity based on a combination of grids and hashing techniques. Although offering strong privacy guarantees, these schemes usually involve complicated algorithms. Moreover, for many services, storing the actual location of their users is critical due to data monetization. For these reasons, these systems have not been adopted by current location services. On the other hand, our approach draws the most fundamental elements from the systems described earlier in order to provide a minimum level of privacy for the users. Our approach can also be implemented using a very simple numeric transformation on the coordinates of the user, making it attractive for adoption in practice.

With respect to *Location Privacy*, Gruteser and Grunwald [2003] introduce spatiotemporal cloaking for providing coarse-grained information to a service, while retaining functionality. Shokri et al. [2011] provide a formal framework for quantifying the privacy guarantees of location privacy protection mechanisms, and a model for formulating location information disclosure attacks. In follow-up work [Shokri et al. 2012], they present a framework that enables a service designer to find the optimal privacy preservation mechanism tailored to each user's desired level of privacy and quality of service. Their model is designed for passive adversaries, while we focus on active adversaries that interact with the service. With respect to the current work, Andrés et al. [2013] present a technique for adding Laplacian noise to a user's location to achieve geo-indistinguishability; in their follow-up work [Bordenabe et al. 2014], the authors use linear programming to maximize utility while achieving geo-indistinguishability. The aforementioned lines of work can be considered as alternatives to the solutions presented in this article if a service wishes to offer stronger privacy guarantees to its users.

A series of works propose *Location Verification* schemes, which are orthogonal to the grid-based scheme considered in this work. Narayanan et al. [2011] propose location tags for proximity verification protocols. Lin et al. [2012] explore the use of GSM cellular networks for creating location tags. While a promising approach, it presents significant limitations prohibiting adoption; users must belong to the same cellular provider, and the maximum permissible distance is too restricting. Marforio et al. [2014] propose the use of smartphones as part of a location verification scheme for sales transactions. A similar approach could potentially be employed by services for verifying users' locations.

Prior to our work, there have been a series of *attacks disclosing users' location*. Foo Kune et al. [2012] demonstrated an attack at the GSM level that could verify if a user is present within a small region. This, however, requires special hardware and could not be employed in the context of location verification required for proximity functionality. Mascetti et al. [2013] present an attack against users that obfuscates their location with dummy queries, which uses clustering, population density data, and trilateration, to bound users within an area. The attack works only for users located in large cities and under unrealistic assumptions of uniform user density. Even then, the bounding area is too large to locate users or infer sensitive data. Recent work [Feng and Shin 2014; Qin et al. 2014] has demonstrated straightforward trilateration attacks against

services that return exact distances. Finally, an independent recent study, Li et al. [2014], presents an attack against proximity services. The attacks presented in this work lack the systematic formality of the current work and fail to eliminate even simple defenses deployed by the services. For instance, the attacks presented against rounded distances contain an average error of 130m for Skout, even before the service introduced noise, and 51m for WeChat; in most cases, their results are too coarse-grained to physically locate a user. The duration of the attacks is also prohibitive for most realistic scenarios, as they require 6.5min to 30min for completion. For moving users, the attack's accuracy is calculated based on the last known location of the user, which is updated every 30min, introducing significant error in the calculation. Furthermore, when attempting to identify users' favorite locations, they consider an identification successful if the estimated locations are within 100m (WeChat) or 0.5mi (Skout) of the actual location. Despite the very-coarse grain assumptions, many of users' top locations remain unidentified after 3wk of monitoring.

10. CONCLUSIONS

In this article, we explored the privacy guarantees of location proximity services. We formalized the problem of user discovery attacks, and designed algorithms with query complexities that match the proven lower bound within a constant factor. We systematically evaluated four popular services and identified the practical challenges of deploying user discovery attacks in the wild. Even though services have deployed countermeasures to prevent trilateration, our experiments demonstrated that they remain vulnerable to more advanced attacks. Based on the insight obtained from the experiments, and the characteristics of our attacks, we proposed a set of guidelines for implementing spatial cloaking in proximity services, which ensures a minimum level of privacy. Disclosure of our findings and guidelines led to the adoption of spatial cloaking by major social networks. Thus, we have made a concrete contribution to user privacy, which resulted in the protection of users against the very effective and efficient class of attacks that we have demonstrated. However, many popular apps remain vulnerable, exposing their users to significant threats. As such, we have released an auditing framework for assisting developers and researchers in evaluating the privacy offered by proximity services.

APPENDIXES

A. EXPECTED DISTANCE LEMMA

LEMMA A.1. *Let G be a square with edge length ℓ and consider the centroid p_c of G. Let $p_r \in \mathbb{R}^2 \cap G$ selected uniformly at random. Then,*

$$\mathbb{E}[\text{dist}(p_r, p_c)] \approx 0.3825978 \cdot \ell$$

PROOF. Consider a square of side ℓ that, without loss of generality, is centered at $(0, 0)$. Let $p_r = (X, Y)$, where X, Y are random variables and let $D(x, y) = \sqrt{x^2 + y^2}$. The distance of p_r from the centroid $p_c = (0, 0)$ of the square is given by

$$\text{dist}(p_c, p_r) = D(X, Y) = \sqrt{X^2 + Y^2}$$

If $d(x, y)$ is the probability mass function for $D(x, y)$, since the random variables X, Y are independent, it holds that

$$d(x, y) = d(x)d(y) = (1/\ell)(1/\ell) = \frac{1}{\ell^2},$$

where $d(x), d(y)$ are the probability mass functions for the distances on the x and y axes, respectively. Therefore, the expected value of the distance is

$$\mathbb{E}[\text{dist}(p_c, p_r)] = \int_{-\ell/2}^{\ell/2}\int_{-\ell/2}^{\ell/2} d(x,y)D(x,y)\,dx\,dy = \int_{-\ell/2}^{\ell/2}\int_{-\ell/2}^{\ell/2} \frac{1}{\ell^2}\sqrt{x^2+y^2}\,dx\,dy \approx 0.3825978 \quad \square$$

B. PROOF OF THEOREM 3.6

PROOF. We will start by proving a number of useful lemmas:

LEMMA B.1. *Let S be a set of points, k be the short edge of $R(S)$ and p, p' two successive points on l_m. Furthermore, let $S_1 = S \cap C_r(p)$ and $S_1' = S \cap C_r(p')$. Then,*

$$|S_1'| \leq |S_1| + O(k)$$

PROOF. The number of points that are added into the disk by shifting the cut by one point are, at most, proportional to the length l of the arc segment with chord length k. We will show that the length is $O(k)$.

First, note that the length of the arc is less than the perimeter of the circumscribed rectangle around the arc segment. Therefore, if h is the height of the arc segment, then we have that $l \leq 2k + 2h$. We want to bound the height h to be less than k. Therefore, we have that

$$h \leq k \Leftrightarrow r - \sqrt{r^2 - k^2/4} \leq k \Leftrightarrow k \leq \frac{8r}{5}$$

where, in the second inference, we used the fact that the height of an arc segment is $r - \sqrt{r^2 - k^2/4}$. Thus, If $k > \frac{8r}{5}$, we have that $l \leq \pi \cdot r \leq O(k)$. On the other hand, if $k \leq \frac{8r}{5}$, we have that $l \leq 2k + 2h \leq 4k \leq O(k)$; therefore, the lemma holds trivially. $\square$

To facilitate computations, we work with the circumscribed rectangles of the sets that result from the cuts of the algorithm. The next lemma shows that the number of points within the sets resulting from cuts and the number of points in their circumscribed rectangles are of the same order.

LEMMA B.2. *Let S be the input set in any iteration of the algorithm. Then, we have that $|S| = \Theta(|I_{R(S)}|)$.*

PROOF SKETCH. We will show that if the point sets that are input to the algorithm satisfy the invariant before the cut, then the new sets will also satisfy the invariant.

Base Case: The first input is all the points in a disk, which is of the order of $\pi \cdot r^2$. The circumscribed polygon (in this case, a square) contains $(2r+1)^2$ points; therefore, clearly in this case, we have that $|S| = \Theta(I_{R(S)})$.

Inductive Step: Assume now that we are at some iteration where the input set satisfies the invariant. Note that, by definition, for all sets S, $|S| = O(|I_{R(S)|})$. Thus, we only have to show that, after the cut, we will have that, for the new set S, $|S| = \Omega(|I_R S|)$.

Let S_1, S_2 be the two subsets after the cut and furthermore assume, without loss of generality, that $|S_1| > |S_2|$. Then, the following hold:

(1) $|I_{R(S_1)|} \leq |I_{R(S)}| \leq O(|S|)$.
(2) $|S_1| \geq |S|/2 \geq \Omega(|S|)$.

It follows from (1) and (2) that $|S_1| = \Omega(|S|)$ and $|I_{R(S_1)}| = O(|S|)$. Thus, we have that $|S_1| = \Omega(|I_{R(S_1)}|)$, which concludes the proof for S_1. The proof for the set S_2 is similar and, thus, omitted. $\square$

The next lemma states a bound on the size of the cuts performed by the algorithm.

LEMMA B.3. *Let $S_l \in \{S_1, S_2\}$ be the largest subset of S after a cut of the algorithm. Then, we have that $|S|/2 \leq |S_l| \leq |S|/2 + O(\sqrt{|S|})$.*

PROOF. Consider the short edge of $R(S)$ with length k. Then, $|I_{R(S)}| = \Omega(k^2)$. Moreover, as proven in Lemma B.2, we have that $|S| = \Theta(|I_{R(S)}|)$; thus, $k = O(\sqrt{|S|})$. Now, consider the largest set S_l of the cut. By Lemma B.1, we have that

$$|S|/2 \leq |S_l| \leq |S|/2 + O(k) \leq O(\sqrt{|S|}),$$

which concludes the proof. □

We now proceed with the proof of Theorem 3.6: Note that since our initial set S contains all the points in a disk of radius r, the number of points that one should test during the linear sweep is at most r at each iteration. In addition, since each iteration makes one query to the proximity oracle, all that remains is to bound the number of iterations of the algorithm.

First, we note that each time we divide S into S_1, S_2, we have that $|S_1| < |S|$ and $|S_2| < |S|$. This is because the algorithm performs a linear sweep in parallel to the long edge of the circumscribed polygon of $|S|$. For every set S such that $|S| > 1$, we will have that the long edge of $I_{R(S)}$ will have length $l > 1$. Therefore, it will always be the case that more than one point along the line l_m will be cutting S. Therefore, both sets S_1, S_2 will contain at least one point and the algorithm will eventually terminate.

Using Lemma B.3, we can bound the total number of queries as follows. We have that queries of the algorithm are bounded by the recursive equation $T(n) = T(n/2 + K(n)) + 1$, for some function $K(n)$. Note now that, because $K(n) \leq O(\sqrt{n})$, there exists an integer c such that, for all $n > c$, we have that $K(n) \leq n/4$. Therefore, for all $n > c$, we have that $T(n) \leq T(n/2 + n/4) + 1 \leq O(\log n)$. When $n \leq c$, the algorithm would need a constant number of additional queries. Therefore, the total number of queries to the proximity oracle is at most $O(\log n)$, which concludes Theorem 3.6.

C. PROOF OF THEOREM 3.7

We prove the theorem by first showing a lower bound for the Disk Search problem.

LEMMA C.1. *Any algorithm solving the Disk Search problem has to make $\Omega(\log r)$ queries to the proximity oracle $P_{r,u}(\cdot)$.*

PROOF. Consider the binary decision tree of the algorithm based on the results of the proximity oracle queries. Then, because there are $\Omega(r^2)$ possible points where the user might reside, the tree must have $\Omega(r^2)$ leaves; therefore, the height of the tree must be at least $\Omega(\log r)$. □

Now, we proceed with the proof of Theorem 3.7.

PROOF OF THEOREM 3.7. In the case that $OPT_{MDS} \geq \Omega(\log r)$, we have that, in the worst case, at least OPT_{MDS} queries will be needed in order to cover the area and we are done. In the opposite case, that $OPT_{MDS} \leq O(\log r)$, one has to at least search a circle of radius r which as, we showed earlier, cannot be done in less than $\Omega(\log r)$ queries.

D. PROOF OF THEOREM 3.11

We prove two lemmas from which Theorem 3.11 follows as a direct implication.

LEMMA D.1. *Let $P_{S,u}(\cdot)$ be a rounding proximity oracle indexed by a family of rounding classes $S = \{R_1, R_2, \ldots, R_l\}$ and an area A in the discrete Euclidean plane. Then, any*

deterministic algorithm solving the Rounding User Discovery Problem in the area A given access to $P_{S,u}(\cdot)$ requires $\Omega(\log \delta_1)$ queries to $P_{S,u}(\cdot)$ in the worst case.

PROOF. Let $S = \{(\mathbb{R}^+, \delta)\}$ and let A be a disk of radius $r = \delta/2 - \epsilon$, for some ϵ, $0 < \epsilon < \delta/4$. We will show that the rounding oracle $P_{S,u}(\cdot)$ can be simulated by an oracle that outputs only one bit of advice on each query. Afterwards, a straightforward application of the decision-tree technique allows us to obtain a $\Omega(\log \delta)$ lower bound on the number of queries required to locate the user since the area contains at least $\Omega(\delta)$ points.

More formally, consider a query $P_{S,u}(p)$ on a point p. Denote by p_c the center point of the disk of the area A. Then, by the triangle inequality, we have that

$$\text{dist}(p, p_c) - r \leq \text{dist}(p, p_u) \leq \text{dist}(p, p_c) + r.$$

Note that, within the interval $[\text{dist}(p, p_c) - r, \text{dist}(p, p_c) + r]$, there exists at most one integer m such that $\delta \mid m$. We define an oracle $O_u(p)$ as follows:

$$O_u(p) = \begin{cases} 0 & \text{if } \lceil \text{dist}(p, p_u) \rceil_\delta = \lceil \text{dist}(p, p_c) - r \rceil_\delta. \\ 1 & \text{Otherwise} \end{cases}$$

Now, the rounding oracle $P_{S,u}(\cdot)$ can be simulated as follows:

$$P_{S,u}(p) = \lceil \text{dist}(p, p_c) - r \rceil_\delta + O_u(p) \cdot \delta.$$

Now, we can use the same argument as in Lemma C.1: the binary decision tree based on the answers of the oracle $O_u(\cdot)$ must have $\Omega(\delta)$ leaves and therefore a height of at least $\Omega(\log \delta)$. □

Lemma D.1 is interesting in the sense that it demonstrates that after the area is reduced into the order of the smallest rounding value, the rounding proximity oracle answers provide at most one bit of additional information in each query. Our next lemma states a lower bound on the queries required to reduce the distance to p_u into the interval of the smallest rounding class.

LEMMA D.2. *Let $P_{S,u}$ be a rounding proximity oracle indexed by a family of rounding classes $S = \{(I_1, \delta_1), \ldots, (I_n, \delta_n)\}$ and an area A in the discrete Euclidean plane. Then, any algorithm for finding a point $p \in A$, such that $\text{dist}(p, p_u) \in I_1$ requires at least $|S| - 1$ queries to $P_{S,u}$ in the worst case.*

PROOF SKETCH. Let S be a rounding class family of size n such that, for any rounding value $\delta_i, i > 1$, we have that $\delta_i \in I_{i-1} \wedge \delta_i > \pi \cdot (\inf I_{i-1} + \delta_{i-1})$. Moreover, consider an area A, where $|A| > \pi \cdot (\inf I_n + \delta_n)^2$. We will show that any problem instance in which the set S and the area A satisfy these constraints requires at least $|S| - 1$ queries to $P_{S,u}$ in the worst case. The proof is by induction on the size of S.

Base case: For $|S| = 2$, since $|A| > \pi \cdot (\inf I_2)^2$, at least one query is required in the worst case.

Inductive step: Now, let us assume that for every set S of size at least k, any algorithm given as input an area A with size $|A| > \pi \cdot (\inf I_k + \delta_k)^2$ requires at least $k - 1$ queries to the proximity oracle $P_{S,u}$.

Consider now a family S, with $|S| = k + 1$ and a single query of the algorithm at any point within an area of size $|A| \geq \pi \cdot (\inf I_{k+1} + \delta_{k+1})^2$. Then, note that, for any query of the algorithm on a point p, there exists at least one candidate point for the user such that $\text{dist}(p_u, p) > \inf I_{k+1}$; therefore, the candidate area for p_u is of size at least $\delta_{k+1}^2 > (\pi \cdot \inf I_k + \delta_k)^2 > \pi \cdot (\inf I_k + \delta_k)^2$. Thus, the algorithm now has to search an area A', where $|A'| > \pi \cdot (\inf I_k + \delta_k)^2$ with the family of rounding classes $S_k = S \backslash \{(I_{k+1}, \delta_{k+1})\}$. By the inductive hypothesis, this requires at least $k - 1$ queries. □

REFERENCES

Miguel E. Andrés, Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. 2013. Geo-indistinguishability: Differential privacy for location-based systems. In *ACM CCS'13*.

Leyla Bilge, Thorsten Strufe, Davide Balzarotti, and Engin Kirda. 2009. All your contacts belong to us: Automated identity theft attacks on social networks. In *ACM WWW'09*.

Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. 2014. Optimal geo-indistinguishable mechanisms for location privacy. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 251–262.

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, New York, NY.

A. Chaabane, G. Acs, and M. A. Kaafar. 2012. You are what you like! Information leakage through users' interests. In *NDSS'12*.

Brent N. Clark, Charles J. Colbourn, and David S. Johnson. 1990. Unit disk graphs. *Discrete Mathematics* 86, 1–3, 165–177.

Josh Constine. 2014a. Techcrunch - Ambient Proximity Is The Next Phase Of Location Sharing. Retrieved November 28, 2016 from http://techcrunch.com/2014/05/01/ambient-proximity.

Josh Constine. 2014b. Techcrunch - Facebook Launches Nearby Friends With Opt-In Real-Time Location Sharing To Help You Meet Up. Retrieved November 28, 2016 from http://techcrunch.com/2014/04/17/facebook-nearby-friends.

Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. 2001. *Introduction to Algorithms* (2nd ed.). McGraw-Hill Higher Education, New York, NY.

Minati De, Gautam K. Das, and Subhas C. Nandy. 2011. Approximation algorithms for the discrete piercing set problem for unit disks. In *CCCG*.

Federal Communications Commission. 2015. Wireless E911 location accuracy requirements. Ps Docket 07-114 (2015).

Huan Feng and Kang G. Shin. 2014. POSTER session: Positioning attack on proximity-based people discovery. In *CCS'14*.

D. Foo Kune, J. Koelndorfer, N. Hopper, and Y. Kim. 2012. Location leaks on the GSM air interface *(NDSS'12)*.

Police Forum. 2013. Police Forum - Social Media and Tactical Considerations For Law Enforcement. Retrieved November 28, 2016 from http://www.policeforum.org/assets/docs/Free_Online_Documents/Technology/social%20media%20and%20tactical%20considerations%20for%20law%20enforcement%202013.pdf.

Sam Frizell. 2014. Time - Tinder Security Flaw Exposed Users' Locations. Retrieved November 28, 2016 from http://time.com/8604/tinder-app-user-location-security-flaw/.

Gabriel Ghinita, Maria Luisa Damiani, Claudio Silvestri, and Elisa Bertino. 2009. Preventing velocity-based linkage attacks in location-aware applications. In *GIS'09*.

Glenn Greenwald and Ewen MacAskill. 2013. The Guardian - NSA Prism program taps in to user data of Apple, Google and others. Retrieved November 28, 2016 from http://www.guardian.co.uk/world/2013/jun/06/us-tech-giants-nsa-data.

Grindr. 2014. Grindr - Location Security Update. Retrieved November 28, 2016 from http://grindr.com/blog/grindrs-location-security-update/.

Marco Gruteser and Dirk Grunwald. 2003. Anonymous usage of location-based services through spatial and temporal cloaking. In *MobiSys*.

Tanzima Hashem, Lars Kulik, and Rui Zhang. 2003. Countering overlapping rectangle privacy attack for moving kNN queries. *Information Systems* 38, 3.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2001. *The Elements of Statistical Learning*. Springer, New York, NY.

Ming-Shih Huang and Ram M. Narayanan. 2014. Trilateration-based localization algorithm using the Lemoine point formulation. *IETE Journal of Research* 60, 1, 60–73.

Yaoqi Jia, Xinshu Dong, Zhenkai Liang, and Prateek Saxena. 2014. I know where you've been: Geo-inference attacks via the browser cache. In *W2SP'14*.

Richard Lardner. 2010. Huffington Post - Feds Using Fake Online Profiles To Spy On Suspects. Retrieved November 28, 2016 from http://www.huffingtonpost.com/2010/03/16/fbi-uses-fake-facebook-pr_n_500776.html.

Hong Ping Li, Haibo Hu, and Jianliang Xu. 2013. Nearby friend alert: Location anonymity in mobile geosocial networks. *IEEE Pervasive Computing* 12, 4, 62–70.

Muyuan Li, Haojin Zhu, Zhaoyu Gao, Si Chen, Le Yu, Shangqian Hu, and Kui Ren. 2014. All your locations belong to us: Breaking mobile social networks for automated user location tracking. In *MobiHoc*.

Zi Lin, Denis Foo Kune, and Nicholas Hopper. 2012. Efficient private proximity testing with GSM location sketches. In *Financial Cryptography and Data Security*.

M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. 1995. Simple heuristics for unit disk graphs. *NETWORKS* 25.

Claudio Marforio, Nikolaos Karapanos, Claudio Soriente, Kari Kostiainen, and Srdjan Capkun. 2014. Smartphones as practical and secure location verification tokens for payments *(NDSS'14)*.

Donald W. Marquardt. 1963. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics* 11, 2, 431–441.

Sergio Mascetti, Letizia Bertolaja, and Claudio Bettini. 2013. A practical location privacy attack in proximity services. In *MDM*. IEEE.

Sergio Mascetti, Claudio Bettini, Dario Freni, Xiaoyang Sean Wang, and Sushil Jajodia. 2009. Privacy-aware proximity based services. In *MDM*. IEEE.

Sergio Mascetti, Dario Freni, Claudio Bettini, X. Sean Wang, and Sushil Jajodia. 2011. Privacy in geo-social networks: Proximity notification with untrusted service providers and curious buddies. *The VLDB Journal The International Journal on Very Large Data Bases* 20, 4, 541–566.

Shigeru Masuyama, Toshihide Ibaraki, and Toshiharu Hasegawa. 1981. Computational complexity of the m-center problems on the plane. *IEICE Transactions* E64, 2, 57–64.

Kazuhiro Minami and Nikita Borisov. 2010. Protecting location privacy against inference attacks *(WPES'10)*.

Arvind Narayanan, Narendran Thiagarajan, Michael Hamburg, Mugdha Lakhani, and Dan Boneh. 2011. Location privacy via private proximity testing. In *NDSS'11*.

Tim Nieberg and Johann Hurink. 2006. A PTAS for the minimum dominating set problem in unit disk graphs. In *WAOA*.

Rick Noack. 2014. Washington Post - Could using gay dating app Grindr get you arrested in Egypt? RetrievedNovember 28, 2016 from http://www.washingtonpost.com/blogs/worldviews/wp/2014/09/12/could-using-gay-dating-app-grindr-get-you-arrested-in-egypt/.

Callum Paton. 2014. The Independent - Grindr and Egypt. Retrieved November 28, 2016 from http://www.independent.co.uk/news/world/africa/9757652.html.

Iasonas Polakis, Stamatis Volanis, Elias Athanasopoulos, and Evangelos P. Markatos. 2013. The man who was there: Validating check-ins in location-based services. In *ACSAC'13*.

Krishna P. N. Puttaswamy and Ben Y. Zhao. 2010. Preserving privacy in location-based mobile social applications *(HotMobile'10)*.

Guojun Qin, Constantinos Patsakis, and Mélanie Bouroche. 2014. Playing hide and seek with mobile dating applications. In *IFIP SEC'14*.

Justin Scheck. 2010. WSJ - Stalkers Exploit Cellphone GPS. Retrieved November 28, 2016 from http://online.wsj.com/articles/SB10001424052748703467304575383522318244234.

Reza Shokri, George Theodorakopoulos, Jean-Yves Le Boudec, and Jean-Pierre Hubaux. 2011. Quantifying location privacy. In *IEEE Security and Privacy'11*.

Reza Shokri, George Theodorakopoulos, Carmela Troncoso, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. 2012. Protecting location privacy: Optimal strategy against localization attacks *(CCS'12)*.

Laurynas Šikšnys, Jeppe Rishede Thomsen, Simonas Saltenis, and Man Lung Yiu. 2010. Private and flexible proximity detection in mobile social networks. In *11th International Conference on Mobile Data Management (MDM'10)*. IEEE, 75–84.

Laurynas Šikšnys, Jeppe R. Thomsen, Simonas Šaltenis, Man Lung Yiu, and Ove Andersen. 2009. A location privacy aware friend locator. In *SST'09*.

George Theodorakopoulos, Reza Shokri, Carmela Troncoso, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. 2014. Prolonging the hide-and-seek game: Optimal trajectory privacy for location-based services. In *WPES'14*.

Federico Thomas and Llus Ros. 2005. Revisiting trilateration for robot localization. *Transactions on Robotics'05* 21, 1.

You-Chiun Wang, Chun-Chi Hu, and Yu-Chee Tseng. 2005. Efficient deployment algorithms for ensuring coverage and connectivity of wireless sensor networks. In *Wireless Internet'05*.

Patrick Wardle. 2014. Synack Security - The Do's and Don'ts of Location Aware Apps; A Case Study. (2014). Retrieved November 28, 2016 from https://www.synack.com/labs/projects/the-dos-and-donts-of-location-aware-apps-a-case-study.

Zheng Yang, Yiyang Zhao, Yunhao Liu, and Yu Xu. 2014. Human mobility enhances global positioning accuracy for mobile phone localization. *IEEE Transactions on Parallel and Distributed Systems* 99, 1.

Xinxin Zhao, Lingjun Li, and Guoliang Xue. 2013. Checking in without worries: Location privacy in location based social networks. In *INFOCOM'13*.

Yao Zheng, Ming Li, Wenjing Lou, and Y. Thomas Hou. 2012. SHARP: Private proximity test and secure handshake with cheat-proof location tags. In *ESORICS'12*.

Ge Zhong, Ian Goldberg, and Urs Hengartner. 2007. Louis, Lester and Pierre: Three protocols for location privacy. In *PETS'07*.

Received December 2015; revised September 2016; accepted October 2016